#! Anatomy of a Bug

Autopsy of CVSS 10.0

The Sandbox Escape

The Patient

- **Name:** Mozilla Firefox
- **CVE:** CVE-2025-2857
- **Diagnosis:** Sandbox Escape
- **Vector:** ipc_channel_win.cc
- **Severity:** 🔥 10.0

What does 10.0 mean?

- **Auth:** None
- **User Interaction:** None
- **Attack Vector:** Remote
- **Result:** Full shell access on the server

#1 The Architecture

The Defense Model

#1.1 The Architecture

The Defense Model

👉 Firefox uses a Multi-Process Model (Electrolysis).

👉 The Parent (Broker): Runs with User Privileges (Medium Integrity). Can touch files, network, devices.

👉 The Content (Renderer): Runs in a Sandbox (Low Integrity). Cannot touch anything.

👉 IPC: The bridge between them. The Parent performs actions on behalf of the Content.

#1.2 The Architecture

Windows Handles

👉 Windows uses Handles to track resources.

👉 A Handle is just an index (integer) in a table.

👉 Process A cannot use Process B's handles directly.

👉 They must be Duplicated (transferred) by the Kernel.

#2 The Vulnerability

Blind Trust in integers

#2.1 The Vulnerability

Blind Trust in integers

👉 **The Flaw:** The IPC logic allows "Relaying" handles.

👉 The attacker (Renderer) sends a message: "Please duplicate Handle X to me".

👉 The Broker executes DuplicateHandle(BrokerProcess, X, RendererProcess...).

👉 The assumption: X is a valid handle index the Renderer gave the Broker.

#2.2 The Vulnerability

The "Magic" Number

👉 The Broker failed to check if X was a Pseudo-Handle.

👉 In Windows:

👉 X = -1 = Current Process.

👉 X = -2 = Current Thread.

👉 The Danger: These values are Context Dependent.

👉 -2 in the Renderer means "Renderer Thread".

👉 -2 in the Broker means "Broker Thread".

#3 The Kill-Chain

Absolute Cinema.

#3.1 The Kill-Chain

The JIT Optimization Trap

👉 The exploit begins when a user visits a website running complex JavaScript.

👉 The JIT (Just-In-Time) Compiler analyzes a specific function that runs thousands of times.

👉 The JIT assumes the data structure is stable (e.g., "always an integer").

👉 It removes safety checks to produce highly optimized machine code.

#3.2 The Kill-Chain

Triggering Type Confusion

👉 The script calls the optimized function one last time with a side-effect.

👉 The code changes the data type from an Integer to an Object in the middle of execution.

👉 The optimized code does not notice the change because the safety checks were removed.

👉 It accesses the memory as an Integer, but it is actually a Pointer.

#3.3 The Kill-Chain

Primitives: addrOf / fakeObj

👉 A type mismatch is now available for exploitation.

👉 addrOf: The payload reads the "Integer" to leak the Memory Address of the Object.

👉 fakeObj: The payload writes a fake "Integer" that the engine treats as a valid JavaScript Object.

👉 A fake ArrayBuffer is constructed pointing to arbitrary memory.

#3.4 The Kill-Chain

Achieving Initial RCE

👉 With the fake ArrayBuffer, Arbitrary Read/Write is achieved over the Renderer Process.

👉 Malicious shellcode is written into a WASM (WebAssembly) page (marked executable).

👉 Control flow is hijacked to jump to this shellcode.

👉 Status: Code execution is achieved within the Renderer, but remains trapped in the Sandbox.

#3.5 The Kill-Chain

The IPC Pivot

👉 To escape, the execution flow pivots to the IPC (Inter-Process Communication) system.

👉 A malicious Relay Message is constructed, destined for the Broker (Parent).

👉 The handle value is set to the magic integer -2 (0xFFFFFFFE).

👉 This integer normally represents the Current Thread in Windows.

#3.6 The Kill-Chain

The Semantic Confusion

👉 The Broker receives the request and calls the Windows API DuplicateHandle.

👉 It passes -2 as the Source Handle to the OS, believing it refers to the Child.

👉 The Windows Kernel sees -2 coming from the Broker Process.

👉 The Kernel resolves it to the Broker's Main Thread.

#3.7 The Kill-Chain

The Fatal Leak

👉 The Kernel creates a THREAD_ALL_ACCESS handle to the Broker and returns it to the requesting process.

👉 The compromised process receives this handle in its Low Integrity context.

👉 It now holds a handle that grants Full Control over the Parent Process.

👉 The Sandbox boundary is effectively destroyed.

#3.8 The Kill-Chain

Weaponization: Thread Hijacking

👉 The payload calls SuspendThread() to freeze the Parent.

👉 GetThreadContext() is used to read the CPU registers.

👉 SetThreadContext() is used to overwrite the Instruction Pointer (RIP).

👉 The RIP is set to point to a ROP Chain or injected shellcode.

#3.9 The Kill-Chain

System Compromise

👉 The payload calls ResumeThread() to unfreeze the Parent.

👉 The Parent Process resumes and immediately executes the malicious payload.

👉 The malware now runs with Medium Integrity (User Privileges), bypassing all restrictions.

👉 Result: Full System Compromise.

#4 The Fix.

Validating the Context

#4.1 The Vulnerable Code

The Broker logic blindly trusted the integer value.

```
// VULNERABLE LOGIC (Conceptual)
// The 'handle_value' comes directly from the untrusted IPC message
HANDLE handle_value = message.ReadInt();

// The Kernel interprets -2 as "Current Thread" (The BROKER'S thread)
DuplicateHandle(
    GetCurrentProcess(), // Source: Broker
    handle_value,        // Value: -2
    target_process,      // Target: Attacker
    &new_handle,
    ...
);
```

#4.2 The Fix

Explicit Rejection

👉 Context Awareness > Type Safety.

👉 We must explicitly forbid the "Magic Numbers" of the OS.

👉 Mozilla imported a helper IsPseudoHandle to check the integer range.

👉 The Security Boundary is restored: Handles are forced to be local indices, never trusted execution contexts.

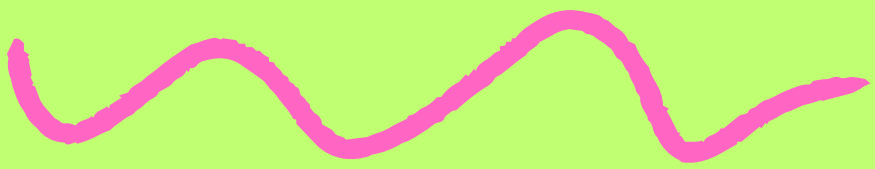
#4.3 The Code Fix

```
// PATCHED LOGIC in ipc_channel_win.cc
// Helper to detect r`magic values` (-1, -2, etc.)
// Windows reserves the range [-12, -1] for g`current context pseudo-handles`.
inline bool IsPseudoHandle(HANDLE handle) {
    int32_t value = (int32_t)(uintptr_t)handle;
    return value >= -12 && value < 0;
}

// In the IPC processing loop:
if (IsPseudoHandle(remote_handle)) {
    // 🔴 STOP: Renderer is trying to trick us.
    LOG(ERROR) << "Security: Received pseudo-handle.";
    return false; // Abort connection
}

// Safe to proceed
DuplicateHandle(..., remote_handle, ...);
```

Status



Discovered by Mozilla.
Patched in March 2025.
Know your input.

#! Anatomy of a Bug

#! Anatomy of a Bug

Technical Credits:

Mozilla Security Team

Author: @tralsesec

#CVE20252857 #ExploitDev #Firefox #SandboxEscape #WindowsInternals #BrowserExploit